iitRACE: A Memory Efficient Engine for Fast Incremental Timing Analysis and Clock Pessimism Removal

Chaitanya Peddawad, Aman Goel, Dheeraj B, Nitin Chandrachoodan

Department of Electrical Engineering
Indian Institute of Technology Madras, India

2015 IEEE/ACM International Conference on Computer Aided Design

Outline

1. Recap
   - Introduction: STA, Incremental Timing and CPPR
   - Problem Formulation
2. Algorithm
   - Incremental Timing: Identifying Incremental Cones and Resolving Dependencies
   - Block-based topologically guided CPPR and Path Extraction Using Dynamic Path Reduction
3. Experimental Results
   - Accuracy and Memory Efficiency
   - Test Coverage and Pin Coverage
   - Challenges & Improvements
4. Conclusion

- Faster turnaround time for timing analysis in presence of design changes
- Clock network as a source of pessimism: Need to update pessimism-free timing information (CPPR) incrementally

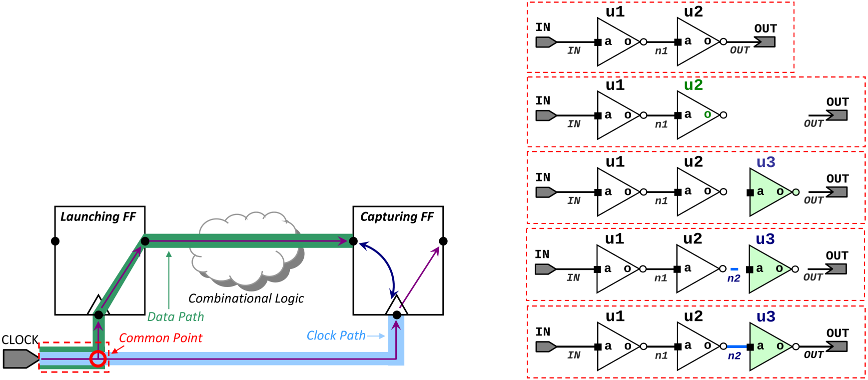


Figure: Example for CPPR and incremental changes to the design

Given a circuit in standard file formats (.v , .lib , .spef , .timing)
The task is to perform incremental changes to the circuit (specified in .ops) and perform timing analysis & CPPR in the *affected regions* using least time and resources

Algorithm
Flow Chart

Perform Incremental Changes

↓

Incremental Timing :
- Identify Incremental Cones
- Resolve Dependencies
- Pre-CPPR Timing Propagation

↓

Block Based CPPR :
- Back Traversal
- Find Credit
- Front Traversal
- Build NegPinList

↓

Path Extraction

↓

Post-CPPR Timing Propagation

Cone-end Points

Every cone in the circuit can be associated with a unique primary output or flip-flop input pin, henceforth referred to as *Cone-end point (CEP)*

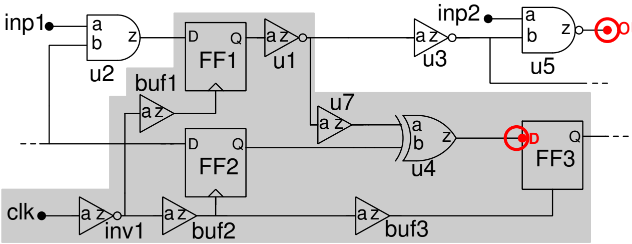


Figure: Cone-end Points: out1, FF3:D

Identifying Incremental CEPs

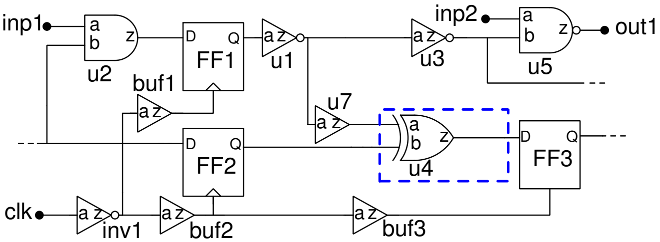


Figure: Adding a gate to the circuit

Identifying Incremental CEPs

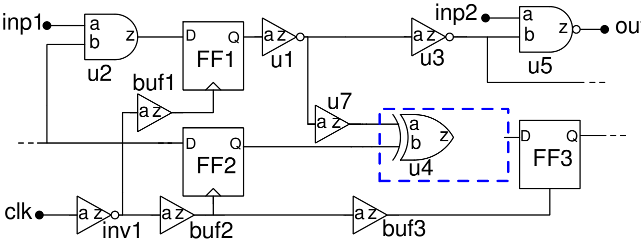


Figure: Adding a gate to the circuit: Disconnect net from u4:z

Identifying Incremental CEPs

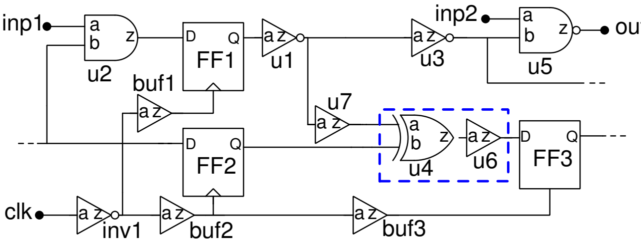


Figure: Adding a gate to the circuit: insert u6

Identifying Incremental Nets

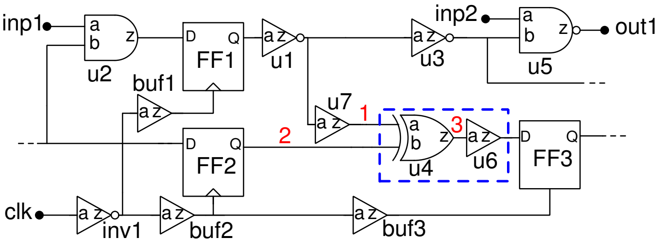


Figure: Adding a gate to the circuit: insert net 3 & connect net 3 to u4:z

Identifying Incremental Nets

- A net and associated timing information at its i/o pins may be dependent on the parameters of another incremental net
- Updating the values is only possible once we resolve the dependencies between incremental nets

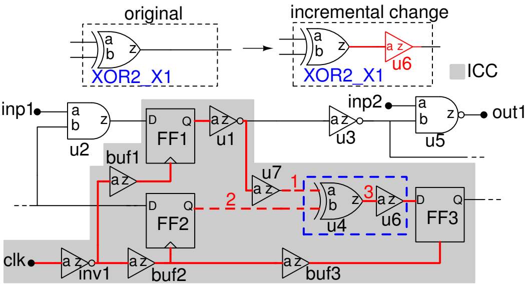


Figure: Incremental Nets

Resolving Dependencies

- Find a set of incrementally affected & independent nets: Based on a modified version of Breadth-First Search Algorithm
- Identify net 1 & 2 as independent nets & FF3:D as incremental CEP
- Cone of FF3:D is hence an incremental cone of change (ICC)

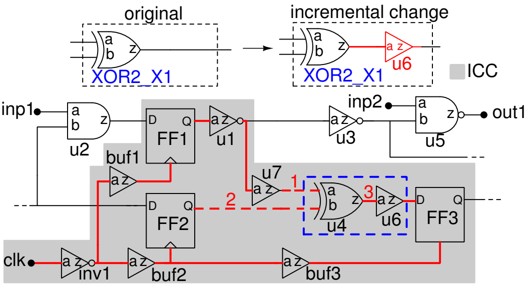


Figure: Nets 1, 2, 3: Incremental Nets 1 & 2: Dependencies resolved

Pre-CPPR Timing Propagation in ICC

- Update AT by single block-based front traversal: start with net 1 & 2
- Update RAT/Slack by back traversal from incremental CEPs (FF3:D)
- Static run (full circuit) vs incremental run (only ICC)

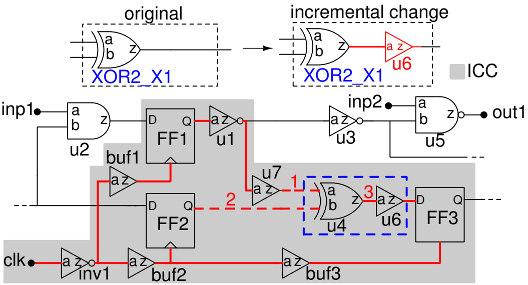


Figure: Timing propagation in ICC

Step 1 - Back Traversal

- Block based levelised back traversal from a CEP till a FF or PI
- Concept of criticalAT & criticalRAT
- Setting RAT (pre-CPPR) and criticalRAT at pins encountered and marking the cone

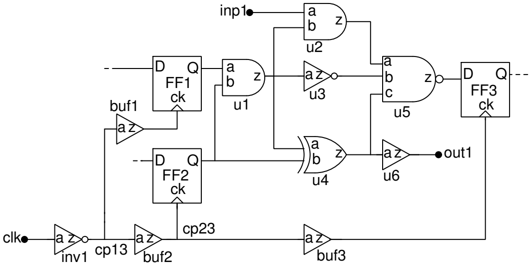


Figure: CPPR Algorithm - Back traversal from FF3:D

Step 1 - Back Traversal

- Block based levelised back traversal from a CEP till a FF or PI
- Concept of criticalAT & criticalRAT
- Setting RAT (pre-CPPR) and criticalRAT at pins encountered and marking the cone

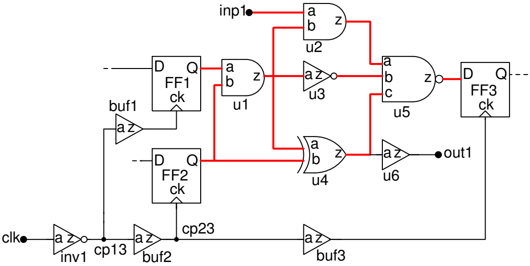


Figure: CPPR Algorithm - Back traversal from FF3:D

Step 2 - Identifying Common Points & Finding Credits

- Identifying common point of data path and clock path for each pair of launching and capturing FFs: cp13, cp23

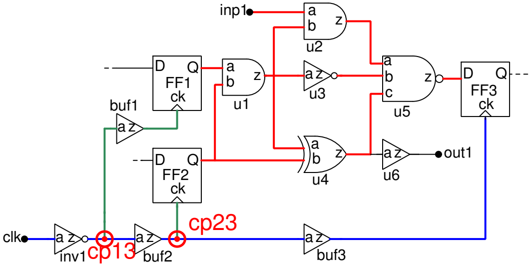


Figure: CPPR Algorithm - Identifying Common Points

Step 2 - Identifying Common Points & Finding Credits

- Credit at a launching FF can be found using eqn -

$$
\begin{aligned}
credit^{hold} &= at^{L}_{cp} - at^{E}_{cp} \\
credit^{setup} &= at^{L}_{cp} - at^{E}_{cp} - (at^{L}_{clk\_src} - at^{E}_{clk\_src})
\end{aligned}
$$

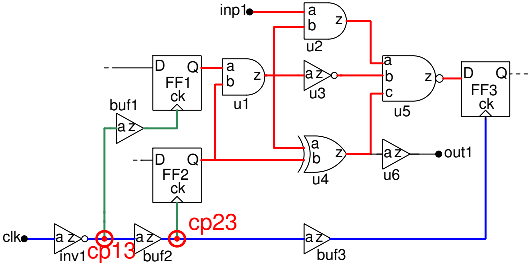


Figure: CPPR Algorithm - Finding Credits

Step 3 - Updating fakeAT

- fakeAT: Adjust AT values to carry credit information at a pin

$$fake\_at_{FF:Q}^{L(E)} \quad = \quad at_{FF:Q}^{L(E)} \mp credit^{L(E)}$$

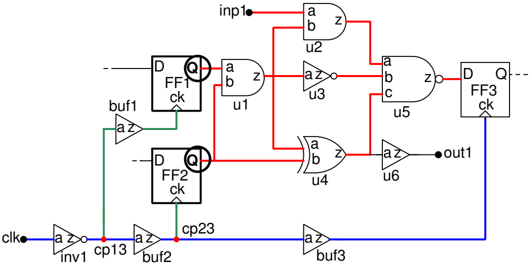


Figure: CPPR Algorithm - Setting fakeAT at output of launching FFs

Step 4 - Front Traversal

- Block-based levelised front traversal within the colored cone
- Propagate fakeAT with setting criticalAT
- fakeAT propagation ensures propagation of worst post-CPPR slack

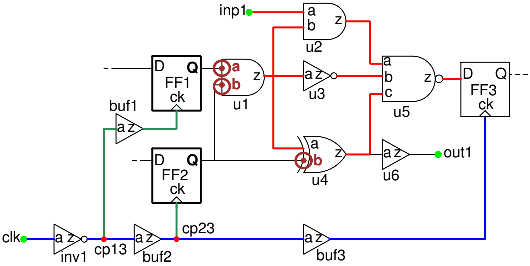


Figure: CPPR Algorithm - Front Traversal

Step 4.1 - Building NegPinList During Front Traversal

- Find the updated slacks using fakeAT and RAT values
- NegPinList & Global Path Table (GPT): Initially empty !
- Add failing pins to NegPinList

NegPinList	
Pins	Slack
$u1{:}a^L$	-33
$u1{:}b^L$	-25
$u4{:}b^L$	-15

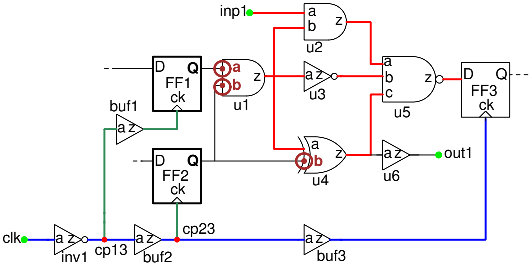


Figure: CPPR Algorithm - Building NegPinList

Step 4.1 - Building NegPinList During Front Traversal

- Find the updated slacks using fakeAT and RAT values
- NegPinList & Global Path Table (GPT): Initially empty !
- Add failing pins to NegPinList

NegPinList	
Pins	Slack
u1:a^L	-33
u1:b^L	-25
u4:b^L	-15
u2:b^L	-28
u2:a^E	-13
u3:a^L	-23
u4:a^L	-33
u5:a^L	-28
u5:a^E	-13
u5:b^L	-23
u5:c^L	-33
FF3:D^L	-33
FF3:D^E	-13

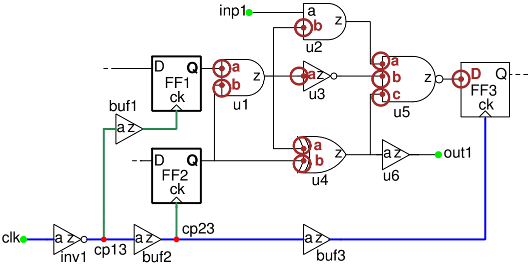
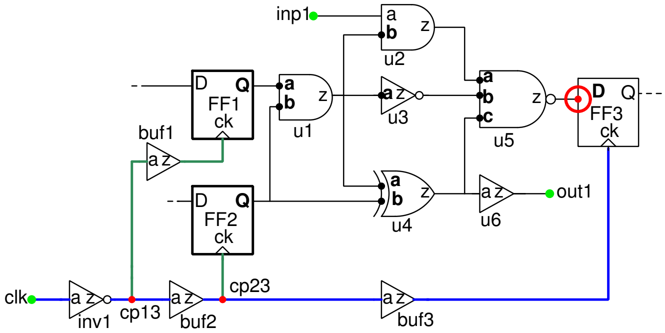


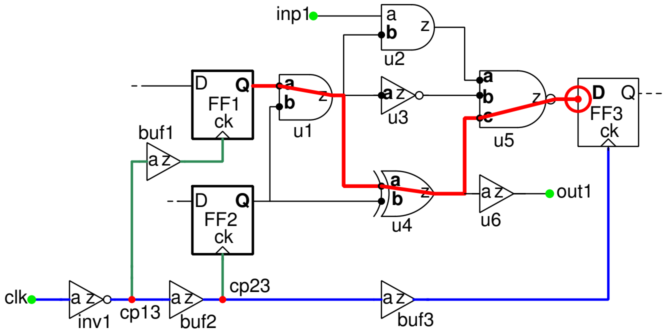
Figure: CPPR Algorithm - Building NegPinList

Step 5 - Extract Paths from NegPinList



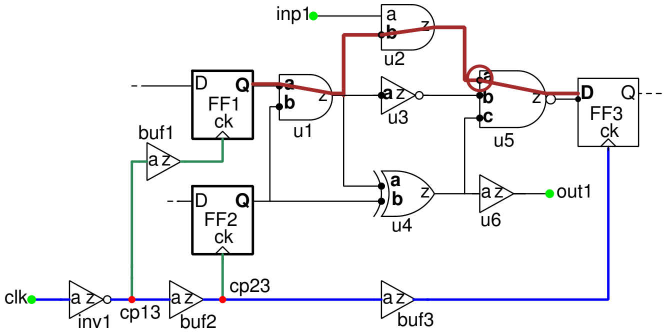
Step	Path	NegPinList
0	-	FF3:D^L, u5:c^L, u4:a^L, u1:a^L, u5:a^L, u2:b^L, u1:b^L, u5:b^L, u3:a^L, u4:b^L, FF3:D^E, u5:a^E, u2:a^E

Step 5 - Extract Paths from NegPinList



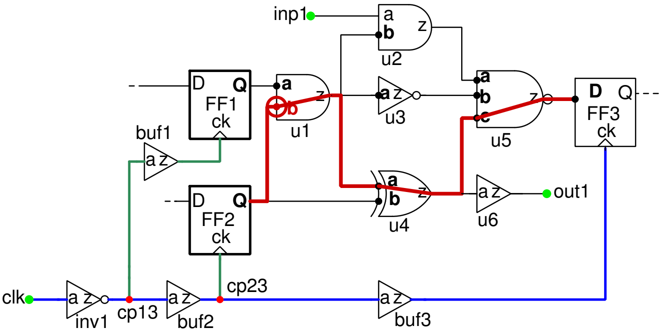
Step	Path	NegPinList
1	P_1	**FF3:D^L**, **u5:c^L**, **u4:a^L**, **u1:a^L**, u5:a^L, u2:b^L, u1:b^L, u5:b^L, u3:a^L, u4:b^L, FF3:D^E, u5:a^E, u2:a^E

Step 5 - Extract Paths from NegPinList



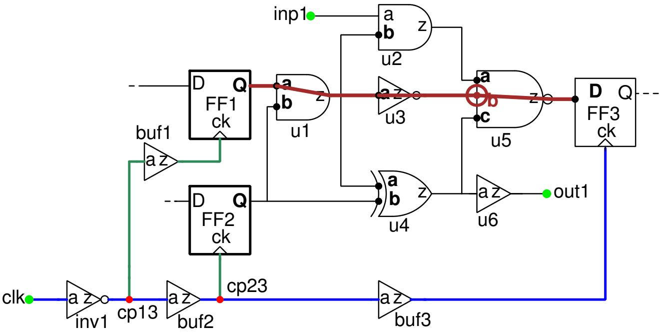
Step	Path	NegPinList
2	P_2	**FF3:D^L**, **u5:c^L**, **u4:a^L**, **u1:a^L**, **u5:a^L**, **u2:b^L**, u1:b^L, u5:b^L, u3:a^L, u4:b^L, FF3:D^E, u5:a^E, u2:a^E

Step 5 - Extract Paths from NegPinList



Step	Path	NegPinList
3	P_3	**FF3:D^L, u5:c^L, u4:a^L, u1:a^L, u5:a^L, u2:b^L, u1:b^L**, u5:b^L, u3:a^L, u4:b^L, FF3:D^E, u5:a^E, u2:a^E

Step 5 - Extract Paths from NegPinList

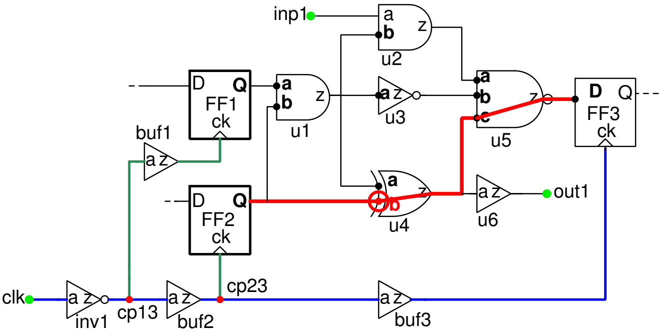


Step	Path	NegPinList
4	P_4	**FF3:D**^{L}, **u5:c**^{L}, **u4:a**^{L}, **u1:a**^{L}, **u5:a**^{L}, **u2:b**^{L}, **u1:b**^{L}, **u5:b**^{L}, **u3:a**^{L}, u4:b^L, FF3:D^E, u5:a^E, u2:a^E

Algorithm

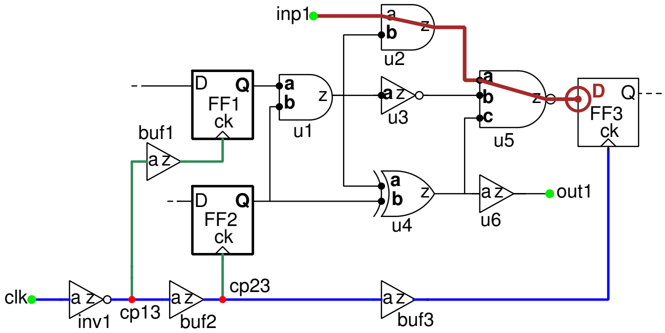
Path Extraction from NegPinList of a cone

Step 5 - Extract Paths from NegPinList



Step	Path	NegPinList
5	P_5	**FF3:D**^{L}, **u5:c**^{L}, **u4:a**^{L}, **u1:a**^{L}, **u5:a**^{L}, **u2:b**^{L}, **u1:b**^{L}, **u5:b**^{L}, **u3:a**^{L}, **u4:b**^{L}, FF3:D^E, u5:a^E, u2:a^E

Step 5 - Extract Paths from NegPinList



Step	Path	NegPinList
6	P_6	**FF3:D^L, u5:c^L, u4:a^L, u1:a^L, u5:a^L, u2:b^L, u1:b^L, u5:b^L, u3:a^L, u4:b^L, FF3:D^E, u5:a^E, u2:a^E**

Paths Skipped

Name	Slack	Mode	Path
P_7	-20	L	FF2:Q→u1:b→u1:z→u2:b→ u2:z→ u5:a→u5:z→FF3:D
P_8	-15	L	FF2:Q→u1:b→u1:z→u3:a→ u3:z→ u5:b→u5:z→FF3:D

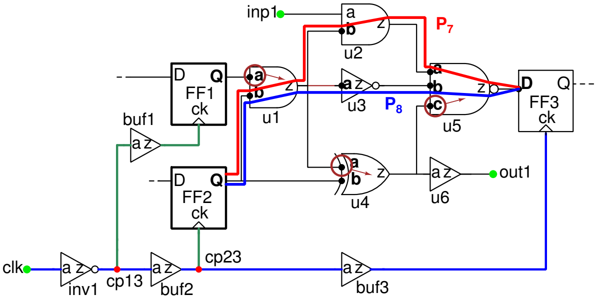


Figure: Path Extraction - Paths skipped

Paths Extraction: Redundant Paths

- None of the pins in path P7 (or P8) have P7 (P8) as worst path through them in the cone under consideration
- It is highly probable that correcting only the reported paths (P1 to P6) would correct the skipped paths (P7 and P8) as well

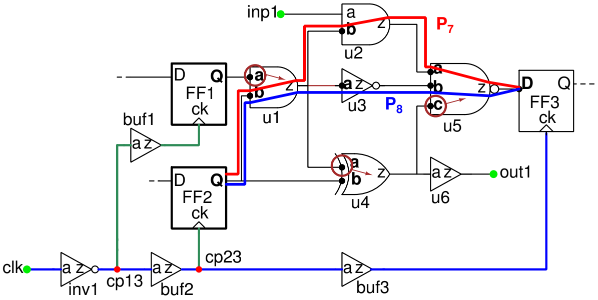


Figure: Path Extraction - Paths skipped

Paths Extraction: Redundant Paths

Most importantly, our proposed algorithm ensures that for every path that is reported, this path is the most critical for some pin in the path, for some logic cone in the circuit. This is not ensured by regular algorithms that report the N worst paths in a circuit, due to which such algorithms typically report many paths that are in some sense **redundant**

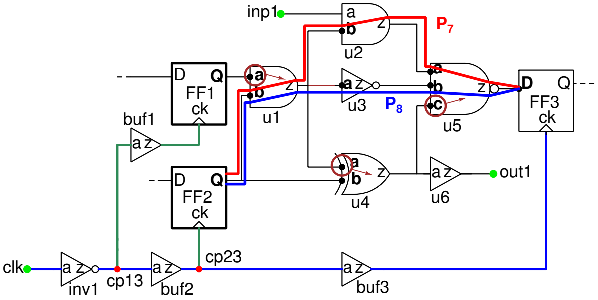


Figure: Path Extraction - Paths skipped

TAU Results: Comparison of iitRACE With Other Academic Timers

Average value accuracy 99% with the least memory requirement !
(Average 2X lower than the first place timer)

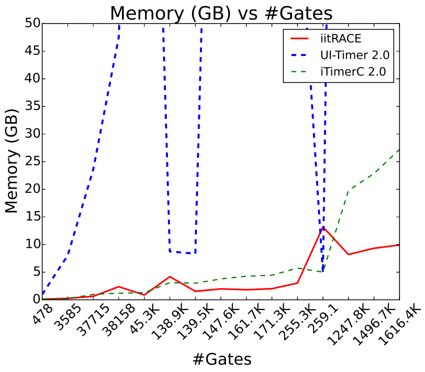


Figure: Memory Usage Comparison

- Memory peaks: corner cases
- On the fly interconnect delay computation
- Pin slack, criticalAT, criticalRAT as the only implicit representation of path

Coverage

- A measure of the number of unique CEPs among the pins in the set of worst paths
- Higher coverage: Our algorithm typically captures a much larger number of such CEPs than the actual N worst paths in the circuit
- Beneficial in identifying all the failing cones

Experimental Results
Test Coverage

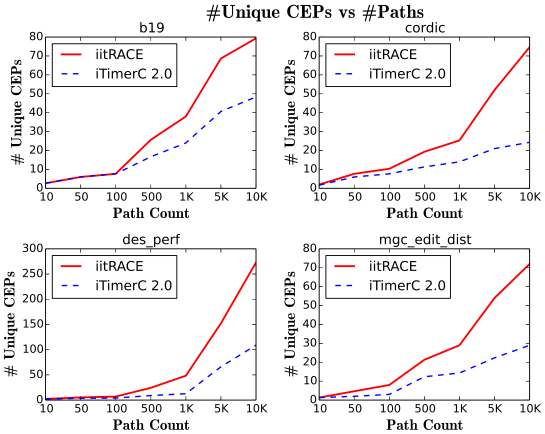


Figure: Test coverage comparison against actual top N worst paths

- Higher coverage of pins: aid to identify critical regions

Post-Contest Speed-up: 10X !

	MMR (GB)		CPU (s)	
Benchmark	**C**	**Post-C**	**C**	**Post-C**
b19	3.02	3.33	426	132
cordic	0.87	0.84	60	31
des_perf	4.19	1.74	189	94
edit_dist	1.98	2.16	562	84
fft	2.38	0.63	44	26
leon2	9.92	12.4	13800	582
leon3mp	8.20	10.17	4920	463
mgc_edit_dist	1.82	2.14	566	79
mgc_matrix_mult	2.01	2.37	239	82
netcard	9.33	11.6	3800	516
tau_cordic_core	0.27	0.21	8	7
tau_crc32d16N	0.11	0.11	1	1
tau_softusb_navre	0.19	0.2	13	7
tau_tip_master	0.63	0.65	39	18
vga_lcd_1	13.22	2.76	742	409
vga_lcd_2	1.54	1.76	243	64
Total	59.68	**53.07**	25680	**2590**

MMR: Maximum Memory Requirement, CPU: Runtime (s)
C: Contest, Post-C: Post-Contest

- Resolved the memory peaks: corner cases
- Cut-off technique: search space reduction
- Algorithmic optimization: sparse table implementation for finding lowest common ancestor (common point), improvement in incremental circuit connection
- Multithreading: parallel processing of the cones

Conclusion

- Proposed a novel memory efficient incremental timing analysis technique with block-based CPPR framework
- The approach is rooted from a highly practical perspective, in which we accurately report only non-redundant critical paths
- Significantly higher coverage of cone-end points corresponding to critical paths than regular algorithms for worst path reporting. This can be used by the designers as an additional aid to identify critical areas in the circuit from a path correction perspective
- **Future extensions**: static/incremental statistical timing analysis

Acknowledgements

- TAU 2015 Contest Organizers
  - Jin Hu, IBM Corp.
  - Greg Schaeffer, IBM Corp.
  - Vibhor Garg, Cadence
- We would also like to thank the authors of UI-Timer 2.0 and iTimerC 2.0 for sharing their timer binaries of TAU 2015 Contest